A neural network closure for the Euler-Poisson system based on kinetic simulations

Leo Bois, Emmanuel Franck, Laurent Navoret, Vincent Vigon

A neural network closure for the Euler-Poisson system based on kinetic simulations

L. Bois[1,2], E. Franck[1,2], L. Navoret[1,2], V. Vigon[1]

August 2020

1. Institut de Recherche Mathématique Avancée, UMR 7501,
Université de Strasbourg et CNRS, 7 rue René Descartes,
67000 Strasbourg, France

2. INRIA Nancy-Grand Est, TONUS Project, Strasbourg, France

Abstract

This work deals with the modeling of plasmas, which are charged-particle fluids. Thanks to machine leaning, we construct a closure for the one-dimensional Euler-Poisson system valid for a wide range of collision regimes. This closure, based on a fully convolutional neural network called V-net, takes as input the whole spatial density, mean velocity and temperature and predicts as output the whole heat flux. It is learned from data coming from kinetic simulations of the Vlasov-Poisson equations. Data generation and preprocessings are designed to ensure an almost uniform accuracy over the chosen range of Knudsen numbers (which parametrize collision regimes). Finally, several numerical tests are carried out to assess validity and flexibility of the whole pipeline.

Contents

1 Introduction

Plasmas are gases composed of charged particles, i.e. ions and electrons, which are the subject of many studies because they are a very common state of matter in the universe (e.g. in stars, ionosphere). They are also present in industrial devices, like in fusion reactors. Plasma dynamics is complex since particles have both long range interactions, through the self-consistent electromagnetic fields, and short-range collisions. The most complete model to describe plasmas is given by the Vlasov equation, which is a kinetic equation satisfied by the distribution function of particles in the position-velocity (x, v)-phase space and which is coupled with the electromagnetic equations. To reduce the dimention, it is possible to use *fluid models* which are derived from the kinetic Vlasov equation with a collision operator by taking the velocity moments corresponding to the monomials 1, v, v^2. This raises up three equations linking the four following quantities: the *density* (the zeroth order moment), the *mean velocity* (involving the two first moments), the *temperature* (involving the three first order moments) and the *heat flow* (involving the four first moments). Fluid models are very attractive as they are much computationally cheaper as they

involve only spatial quantities. However, they require an additional equation called a *closure* to link the heat flow to the other three moments.

The first two fluid models obtained in the collisional regime are the Euler and the Navier-Stokes system. The closure is obtained from an asymptotic Chapman-Enskog procedure, assuming that the distribution function is close to be at thermodynamical equilibrium due to collisions. These equations are thus valid for small Knudsen number $\varepsilon > 0$, which is the mean free path between two collisions divided by the characteristic length studied. The Navier-Stokes system provides $O(\varepsilon)$ corrections of the pressure tensor and heat flux. Higher order corrections have been considered but lead to ill-posed systems (Burnett equations). We refer to [4] for a specific derivation in the case of plasmas and to [10] for a review.

To extend the validity of fluid models for larger Knudsen numbers, larger moment systems have been proposed for the Boltzmann kinetic equation, with neutral particles: the Grad 13 order moment model using perturbative theory [18], the Levermore 14 order moment model using entropy maximisation closure [26, 27]. They both suffer from intrinsic mathematical imperfections: a lack of hyperbolicity for the former [6], so-called realizibility issues for the latter [24, 35]. However, recent developments propose corrections for the Grad model [6] and the entropy methodology and has been very successful in the context of radiative transfer with compact velocity domain [14, 17].

Specific closures have been developed for plasma fluid models. In electrostatic regime, the Hammett-Perkins closure [20, 19] is designed to recover the Landau damping effect, which results from the transfer from spatial modes to velocity modes and which leads to the damping of the electrostatic energy. To this end, the heat flux depends on the temperature through a non-local integral dissipative operator. Dissipation is also introduced in the momentum equation. Let us mention the recent work [28] for a discussion on such closure. When considering magnetized plasma, several closures have also been proposed for fluid models, like for Magnetohydrodynamics (MHD) or gyrofluid equations. See for instance [7, 37, 5, 31]. Recently, fluid closures that preserve the Hamiltonian structure of the Vlasov equation have also been developed [32, 39].

Other strategies consist in adding kinetic effects. Let us mention model reduction technic to obtain fluid models like the water-bag method [2, 3, 12], where the kinetic distribution function is approximated with piece-wise constant functions in velocity. Kinetic effects can also be numerically introduced through micro-macro decomposition methods [9, 11, 8].

A more recent approach consists in using supervised machine learning to find a data-driven closure. Artificial neural networks (ANN) have proven very efficient to interpolate data and detect underlying structures. In particular, convolutional neural networks are very efficient to analyse images and thus the outputs of numerical simulations. There are numerous works for designing physics-based models using neural networks (see for instance [1, 40, 16]). Such tools have already been used in two works concerning moment closure. In [21], in the context of neutral gases, the authors propose to learn the appropriate higher moments required in the model. In [29], which concerns plasmas models,

the closure is directly learned from the analytic closure like the Hamett-Perkins one.

Description of our work. Here, we introduce a data-driven closure, based on a fully-convolutionnal neural network, which is valid for a large spectrum of collisional regimes (i.e. a large range of ε). Data are obtained from numerical simulations of the Vlasov-Poisson system satisfied by the distribution function and the electric potential. This method has already been mentioned in [21] for neutral gases, but the authors preferred to develop another approach. Here we further investigate the design of the closure to obtain accurate predictions.

Collisions between charged particles would normally be modeled with the Fokker-Planck-Landau operator. However, as it is usually done, we replace it by a BGK (Bhatnagar-Gross-Krook) operator, that models the relaxation of the distribution function towards the equilibrium distributions, the Maxwellians. The inverse of the Knudsen number ε is in prefactor of this operator.

Knudsen numbers ε range is chosen equal to $[0.01, 1]$. Indeed, for $\varepsilon < 0.01$, the Navier-Stokes closure already gives good results. An upper bound of the Knudsen interval has to be prescribed. Here we choose $\varepsilon = 1$ for embrassing mildly collisional regimes, but larger values could be considered.

The closure takes as input three one-dimensional vectors corresponding to the spatial discretized density, mean velocity and temperature, and one vector corresponding to the parameter ε. The closure returns an estimation of the spatial discretized heat flux.

The core of the closure is a fully convolutional neural network which has a *V-Net* architecture. It was first described and developed by Milletari et al. [30] to treat medical images (3D signals). It is an evolution of the U-Net, developed for biomedical 2D images [34]. It consists in a succession of several convolution kernels, down-samplings and up-samplings that perform multi-scale analysis of the signal. No fully connected layer is used, so the whole network has relatively few parameters. The architecture of the network can mostly be described by three hyperparameters: the number of levels of the "V", the depth (which rules the number of channels at each level), and the size of the kernels of convolution. The influence of these parameters on the performance is investigated.

As usual in neural networks construction, some processing of the data is required to predict the heat flux in the widest possible range. Therefore multiple steps of processing are included in the closure, on top of the central neural network. One of these is the normalization of the heat flux with the Navier-Stokes approximation, that helps lowering the otherwise large relative error on predictions of low heat fluxes. Another weakens the dependency of the closure on the underlying mesh size, by performing a data slicing before applying the neural network part. We also use a resampling strategy for discretizations with a resolution different from the one used to train the neural network.

The insertion of this closure in a fluid solver raises mathematical issues. Indeed, it is not guaranteed that the neural network closure is dissipative and thus it could lead to instabilities. To prevent such scenario, a smoothing of the

prediction is added into the closure.

The paper is organised as follows. In Section 2, we introduce the Vlasov-Poisson system and the moment closure issue. In particular, we give the Euler and Navier-Stokes closures. Then the neural network closure strategy is explained in Section 3: prediction strategy, architecture of the network and training methodology are presented. Then Section 4 details the data generation and Section 5 their processing. Finally, in Section 6, we carry out several numerical tests to quantify and analyze the accuracy of the closure.

2 Fluid closure for Vlasov-Poisson

In this section we present the kinetic description of the plasma dynamics in one space dimension and its fluid approximations.

2.1 Kinetic model

A kinetic model is a model interested in the function $f : (x, v, t) \mapsto f(x, v, t)$ that describes the evolution of the distribution of the particles in the (x, v)-phase space, where $x \in [0, L]$ denotes the space variable, $v \in \mathbb{R}$ the velocity variable and $t \in \mathbb{R}$ the time. We will consider periodic boundary conditions in space. From this distribution function can be computed some macroscopic physical quantities. The first three moments give the particle density $\rho(x, t)$, the mean velocity $u(x, t)$ and the total energy $w(x, t)$ defined as:

$$\rho(x,t) = \int_{\mathbb{R}} f(x,v,t)dv, \quad \rho(x,t)u(x,t) = \int_{\mathbb{R}} f(x,v,t)vdv, \tag{1}$$

$$w(x,t) = \frac{1}{2}\int_{\mathbb{R}} f(x,v,t)v^2dv. \tag{2}$$

We can also define the pressure $p(x, t)$, the temperature $T(x, t)$ and the heat flux $q(x, t)$:

$$p(x,t) = \int_{\mathbb{R}} f(x,v,t)(v-u(x,t))^2dv, \quad \rho(x,t)T(x,t) = p(x,t), \tag{3}$$

$$q(x,t) = \int_{\mathbb{R}} \frac{1}{2}f(x,v,t)(v-u(x,t))^3dv. \tag{4}$$

Note that we have the following relation: $w = \rho u^2/2 + \rho T/2 = \rho u^2/2 + p/2$. The evolution of the distribution is described by the Vlasov equation:

$$\partial_t f + v\partial_x f - E\partial_v f = Q(f), \tag{5}$$

where $E(x, t)$ is the self-induced electric field, which satisfies the Poisson equation:

$$E = -\partial_x \phi \quad , \quad \partial_{xx}\phi = \rho - \int_{[0,L]} \rho \, dx. \tag{6}$$

Here $\phi(x,t)$ denotes the electric potential.

The source term Q is called a collision operator and allows the model to take into account the collisions between particles. Different collision operators can be considered to deal with different situations. In this work we use the BGK operator (Bhatnagar, Gross and Krook), built to conserve the mass, momentum and kinetic energy of the system, and model the relaxation induced by the collisions toward a local equilibrium distribution $M(f)$. This operator simply reads

$$Q(f) = \frac{1}{\varepsilon}(M(f) - f),$$

where $M(f)$ is called the Maxwellian of f and is given by

$$M(f)(x,v,t) = \frac{\rho(x,t)}{\sqrt{2\pi T(x,t)}}\, e^{-\frac{(v-u(x,t))^2}{2T(x,t)}}, \tag{7}$$

with ρ, u and T the density, mean velocity and temperature associated to f and defined in (1)-(3). The parameter $\varepsilon > 0$ is called the Knudsen number and represents the mean free path between two collisions divided by the characteristic length studied. In the limit $\varepsilon \to 0$, the distribution function is expected to be closed to local equilibria. In this regime, the phase-space dynamics could be infered from the spatial dynamics of its macroscopic moments ρ, u, T.

The kinetic equation (5) can be easily solved numerically in this one dimensional case. Several numerical methods have been proposed in the literature [38, 13]. In this work, we use a Finite Difference/Finite Volume method similar to the one introduced in [33, 23] and described in appendix A. Although it is possible to extend such computations in dimension 2 or 3, the computational cost becomes very prohibitive. We are therefore led to consider fluid models.

2.2 Fluid model

A fluid model of a plasma describes the evolution of its density $\rho(x,t)$, mean velocity $u(x,t)$ and kinetic energy $w(x,t)$, instead of the distribution of its particles $f(x,v,t)$ in a kinetic model. Such models are far less expensive to simulate numerically than the original kinetic model. Indeed, this requires the computation of only three spatial quantities ($3 \times N_x$ unknowns), instead of the full kinetic distribution ($N_x \times N_v$ unknowns), where N_x and N_v stands for the number of discretization points in space and velocity.

Fluid models can be obtained by using the moment method [10]. Formally, it consists in computing the first three moments in velocity of the Vlasov equation, i.e. multiplying Equation (5) by 1, v and $v^2/2$ and then integrating in velocity:

$$\text{for } p = 0,\, 1,\, 2, \quad \int_{\mathbb{R}} v^p \left(\partial_t f + \ v\partial_x f - \ E\partial_v f\right) dv = \int_{\mathbb{R}} v^p Q(f)\, dv.$$

Since the collision operator conserves mass, momentum and energy, the right-

hand sides vanish. Therefore, it results in the following system:

$$\begin{cases} \partial_t \rho + \partial_x(\rho u) = 0, \\ \partial_t(\rho u) + \partial_x(\rho u^2 + p) = -E\rho, \\ \partial_t w + \partial_x(wu + pu + q) = -E\rho u, \end{cases} \tag{8}$$

where p is the pressure and q the heat flux defined in (3)-(4). Note that the pressure p is actually a function of ρ, u, w since we have the following relation:

$$p = 2w - \rho u^2.$$

The electric field E is still given by the Poisson equation (6).

We thus get a system of three equations on the four variables ρ, u, w and q. For it to be closed, we need a fourth equation connecting these four unknowns, called a closure. Usually this closure consists in replacing the true heat flux q with a simplified one $\hat{q}$ given as a function of the other quantities , and that can be written

$$\hat{q} = \mathcal{C}(\varepsilon, \rho, u, T).$$

Note that this closure depend on the physical regime we consider through the parameter ε. Let us also mention here that in higher dimension, the pressure stress tensor is not completely determined by ρ, u and w and an additionnal closure relation is necessary.

In fluid regimes, where most of the kinetic effects can be neglected, two closures are classically considered: the Euler or the Navier-Stokes closures. The Euler closure is valid in regimes where the distribution function is closed to be Maxwellian: $f = M(f) + O(\varepsilon)$. Using this ansatz, we obtain the Euler closure:

$$\hat{q} = 0.$$

When we are interested in regime with $O(\varepsilon)$ deviation from the Maxwellians, $f = M(f) + \varepsilon g + O(\varepsilon^2)$, we get the Navier-Stokes closure:

$$\hat{q} = -\frac{3}{2}\varepsilon\, p\, \partial_x T. \tag{9}$$

Note that it is a $O(\varepsilon)$ correction of the Euler closure. We refer to [10] for more details.

More complex closures have been developped to capture more kinetic effects, in regimes where the disbribution function is more distant from the Maxwellians. Such closures are chosen non local, meaning that the heat flux $q(x)$ at location x does not rely on the values of ρ, u, T and their derivatives at location x only, but on their values on all the domain. Indeed, the true heat flux, given by a third order moment of the kinetic distribution (Eq. (4)), depends on the full distribution in velocity at location x and thus retains non-local information about the dynamics.

The goal of our work is to provide a method to provide a new closure, where the function $\mathcal{C}$ is implemented using a neural network. The next section describes this approach in more details.

3 Closure with a neural network

In this section we introduce the overall principle of our method to build the fluid closure presented in Section 2.2 with a neural network. First we describe the basic functioning of the neural network, before introducing the other operations included in the closure, and that surround the neural network. We end this section with the detailed description of our neural network and of its training.

3.1 Interpolation of the heat flux with a neural network

In this work we are interested in the ability of machine learning to find a function C that maps the Knudsen number, the density, the mean velocity and the temperature of the plasma to its heat flux. Such a mapping cannot be exact as the heat flux is not a function of these four quantities but a function of the particles' distribution in the phase space. Nonetheless, the ability of neural networks to find non obvious patterns and correlations can be used to provide a good approximation of the heat flux, that can then be used as closure for the fluid model described in Section 2.2.

Neural networks enable to build functions relying on many parameters. The obtained closure can be formally written as:

$$\hat{q} = C_{\hat{\theta}}(\varepsilon, \rho, u, T),$$

where $\hat{\theta}$ denotes the set of parameters of the obtained network. Here the closure is chosen to be **non-local**: ρ, u, T as well as q are all spatial discretized quantities. The closure takes as input four vectors or, in the neural network terminology, a 1D signal with four channels corresponding respectively to the Knudsen number (turned into a constant vector), the density, the mean velocity and the temperature at spatial discretization points:

$$X = (\varepsilon, \rho, u, T) \in (\mathbb{R}^{N_x})^4,$$

where N_x is the number of spatial discretization points. The output of the closure is one vector of size N_x

$$Y = C_\theta(\varepsilon, \rho, u, T) \in \mathbb{R}^{N_x},$$

that will correspond to the estimated heat flux $\hat{q} \in \mathbb{R}^{N_x}$.

To find such a function, there are two main steps:

1. First, by setting the *architecture* of the neural network, we define a family of functions C_θ parametrized by $\theta \in \Theta$, where Θ represents the set of all the possible parameters of the neural network.

2. Then, by *training* the neural network, we find a set of parameters $\hat{\theta}$ so as to minimize the error of the neural network predictions on a dataset, for which the true heat flux is known.

The neural network is the core component of the closure C_θ, but it is not its only one. Indeed, for the data to be usable by the neural network and to provide satisfying results, it needs to go through a certain amount of processing. Section 3.2 introduces these other components and their articulation in the closure. Sections 3.3 and 3.4 then describe respectively the architecture and the training of the neural network.

3.2 Detailed composition of the closure

In practice, the closure is used at each iteration in time to compute the heat flux q on all the mesh from the input $(\varepsilon, \rho, u, T)$ on all the mesh. Such a use of a neural network based closure into a numerical scheme requires some work to transform the data handled by the solver into data usable by the network, and vice-versa. This section briefly introduces the different steps involved in the closure and their role in relation to the network.

There are three main processing operations designed to integrate the closure into the numerical scheme. The first one deals with the difference in resolution between the numerical scheme and the data used to train the network, by resampling the input of the network to its training resolution, and its output back to the original resolution. The second one deals with the difference in length between the resampled data of the scheme and the inputs handled by the network, by slicing the data into several pieces to give to the network, and then aggregating the outputs to reconstruct the full heat flux. The third one deals with the stability of the resulting numerical scheme, by smoothing the data from the network to prevent any oscillation to propagate into the numerical scheme. These three steps are part of a whole process, that can be broken downs as follows:

$$C_\theta : \quad X \overset{\text{(Re)+(P)}}{\longmapsto} \tilde{X} \overset{\text{(Sl)}}{\longmapsto} (\tilde{X}_j)_j \overset{(\text{NN}_\theta)}{\longmapsto} (\tilde{Y}_j)_j \overset{\text{(R)}}{\longmapsto} \tilde{Y} \overset{\text{(P')+(Sm)+(Re)}}{\longmapsto} Y.$$

This process is illustrated by figure 1, and the different steps are described below.

1. **Resampling (Re) and pre-processing (P)** The input X of size N_x is resampled with a Fourier method (that relies on the periodicity of the signal) and pre-processed into a signal $\tilde{X}$ of size N'_x. As explained in section 6.3.1, the resampling changes the resolution of the input to match the one used for training and get better results. The pre-processing step that follows is a common step in the use of neural networks that allows to improve the accuracy. It is detailed in section 5.1.

2. **Slicing (Sl)** The inputs are sliced into several pieces $(\tilde{X}_j)_j$ of size N before being handed over to the network, where N is the size of the input the neural network works with.

3. **Neural network (NN_θ)** For each piece of input $\tilde{X}_j$, the network predicts a piece of heat flux $\tilde{Y}_j$, of size N.

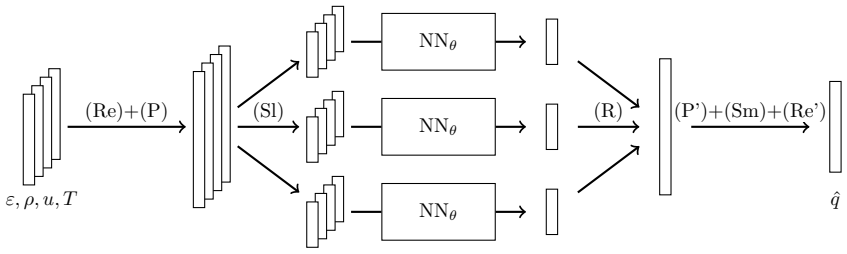

Figure 1: Graphic of the composition of the closure. The different operations are (Re)-(Re') resampling, (P) pre-processing, (Sl) slicing, (NN$_\theta$) neural network, (R) reconstruction, (P') post-processing and (Sm) smoothing.

4. **Reconstructing (R)** The signals $\tilde{Y}_j$ are aggregated in order to reconstruct the whole heat flux. An added benefit of this slicing and reconstructing process is that it allows to prevent some edge effect introduced by the network: by using overlapping pieces, the ends of each piece can be ignored when reconstructing the whole signal. As described in section 5.3, the amount of overlapping is a parameter that can be tweaked.

5. **Post-processing (P'), smoothing (Sm) and resampling (Re')** The post-processing is an operation of normalization designed to improve the accuracy of the network and detailed in section 5.2. The smoothing of the signal is here to avoid oscillations coming from the predictions to propagate and amplify, thus making the numerical scheme unstable. The intensity of this smoothing can be tweaked, and is discussed in section 6.2.4. Finally it is followed by a resampling to recover the original resolution, for the output to be used in the rest of the computations.

These different operations will be fully exposed in Section 5. We now turn to the heart of the closure: the neural network.

3.3 Architecture of the neural network

The closure C_θ depends on the architecture of the network. In our work we use a 1D version of the V-Net [30, 34] implemented with the Tensorflow 2 library through its python interface.

The V-net is a fully convolutional neural network meaning that it is based only on a successive sequence of convolutions (no fully connected layers). It consists of two parts. First a descending part that acts as an "encoder", and decomposes the input X into multiple features, performing a multiscale analysis. Then an ascending part that acts as a "decoder", and synthesizes the created features to predict the output Y. As we will see below, it can be characterized by three hyperparameters: the number of levels ℓ, the depth d of the first convolution and the kernel size p of all convolutions. The composition of the V-net illustrated Figure 2 is the following:

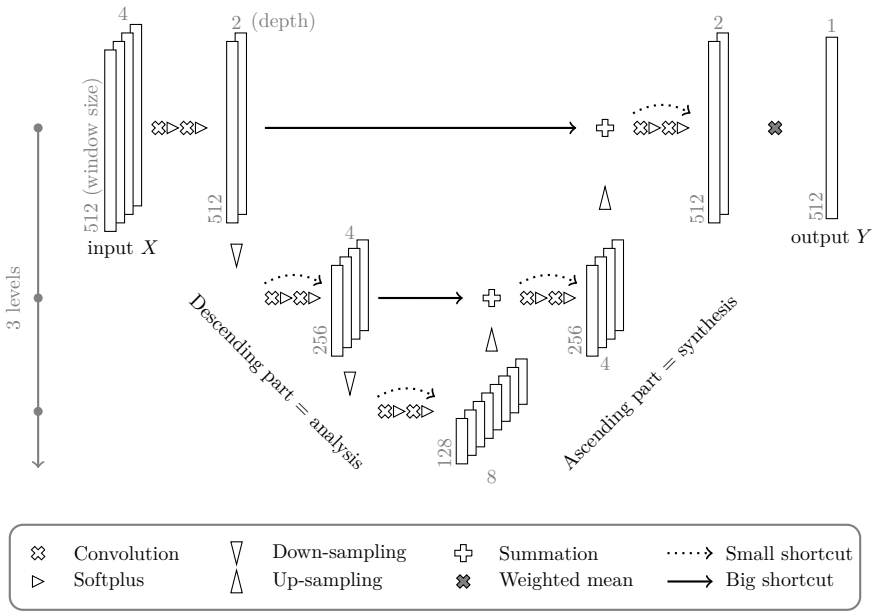

Figure 2: Graphic of a 1D V-Net with a window of size $N = 512$, a depth $k = 2$ and $\ell = 3$ levels.

Input As described previously, it is a 1D signal X made of 4 channels $(\varepsilon, \rho, u, T)$, and with a given length called the *window size* that we set to N.

Initialization We first perform successively two 1D convolutions[1] with a kernel size of p (we tried $p = 5, 7, 9, 11$), both followed by the softplus activation function $s(x) = \ln(1 + e^x)$, and change the number of channels from 4 to a depth d (we tried $d = 4, 5, 6, 8$). All convolutions preserve the size of the signal with the help of a constant padding that extends the signal by continuity. The original V-Net uses the ReLu activation function, but the regularity of the softplus activation function seems to give us better results.

Descending part The V-net is made of ℓ levels (we tried $\ell = 3, 4, 5$), so $\ell - 1$ descents and as many ascents. Let us describe one descent.

- The input is down-sampled by a convolution of kernel size 2, stride 2, and that doubles the depth of its input. The stride allows to halve the length of the input, incidentally blurring the precise locations of the highlighted features.
- Then are applied two successive convolutions of kernel size p and that conserve the depth of their input, both followed by the softplus function.
- The output of this double convolution is added to its input. This "shortcut" is represented by the dotted arrows in Figure 2. This way the convolutions produce additive (or residual) modifications. This is the principle of the famous Resnet. It has been shown that it accelerates the training process and limits the problems of vanishing or exploding gradient [22].

The output of this descent is a signal with half the length and double the depth of the input signal.

Ascending part It works as a mirror of the descending part. Each ascent simultaneously increases the length of the signal and decreases its depth by a factor of 2. Technically, we up-sample the data using a transposed-convolution [15] of size 2 followed by two convolutions of kernel size p with softplus activations.

Long shortcuts At each level, the features obtained in the ascents are combined with the features created by the descents, using a summation. This

[1] For the sake of completeness, we recall the formula for a one dimensional convolution: applied on an input X of shape (N, d) to get an output Y of shape (N, d'), a 1D convolution with an odd kernel size p uses a kernel K of shape (p, d, d') and we have

$$Y_{i,k} = \sum_{j=1}^{d} \sum_{di=1}^{p} \tilde{X}_{i+di,j} K_{di,j,k},$$

where $\tilde{X}$ is the input X padded on both ends so that Yi, k is well defined for i up to N.

allows to retrieve some fine resolution details, which are lost by the blurring produced by down and up samplings.

Weighted mean The resulting signal of depth d is turned into a one dimensional output Y by simply taking a weighted mean of all d channels. This operation is implemented with a convolution of kernel size 1. No activation function is added, as we deal with a regression problem.

All the coefficients of the convolutions constitute the set of parameters θ of the neural network.

Table 1 summarizes the hyper-parameters we choose in practice as a reference. This choice was mostly motivated by the results given in Section 6.1.2. In order to avoid the oscillations that can be produced by the upsampling oper-

Hyper-parameter	Value
size of the input window (N)	512
number of levels (ℓ)	5
depth (d)	4
size of the kernels (p)	11
activation function	softplus

Table 1: Hyper-parameters of the reference neural network

ation of the V-Net architecture, we initialize the transposed convolutions with constant kernels.

3.4 Training of the neural network

The training consists in finding an optimal set of parameters $\hat{\theta}$ for the neural network. It is defined as the minimizer of a loss function that measures the error of the network:

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \ \mathrm{Loss}(\theta).$$

For this loss function we choose the mean absolute error (MAE) between the output of the network and the expected heat flux over a *training dataset*:

$$\mathrm{Loss}(\theta) = \sum_{j \in \text{training dataset}} |\mathrm{NN}_\theta(\tilde{X}_j) - \tilde{Y}_j|,$$

where $(\tilde{X}_j)$ (resp. $\tilde{Y}_j$) are data of size $N \times 4$ (resp. N), obtained from kinetic simulations after pre-processing (P) and slicing (Sl) of vectors $(\varepsilon, \rho, u, T)$ (resp. q). In practice, the minimization is carried out with a gradient descent algorithm.

This whole process falls under the category of supervised learning, as it requires a *labelled* dataset, including both the inputs and the corresponding

expected outputs. The first step to train the neural network is thus to generate such a labelled dataset. To do so, we consider the kinetic model (5)-(6), that describes the particles' distribution from which can be derived their density, mean velocity, temperature, and heat flux. We run many simulations with a given time step and given number of discretization points in space N_x and velocity N_v, with the numerical method described in appendix A. We save the results in our dataset at selected times.

The making of this dataset is described in more details in Section 4. In particular, it requires crucial choices in the initial distributions, the recording times and the range of the parameter ε. It has a direct impact on the range of validity of the closure obtained and it will be numerically discussed in Section 6.

In the end, the network is trained on a dataset of 10 000 entries $(X_k; Y_k) = (\varepsilon_k, \rho_k, u_k, T_k; q_k)$, that are processed as described in Section 5. In particular, each vector (of size $N_x = 1\,024$) is sliced into overlapping windows of size $N = 512$, resulting in 80 000 inputs and labels. 4% of these are isolated as a small test dataset and 10% of what remains are used for validation. The rest is used for the actual training.

This training consists in 5 series of 120 epochs with a decaying learning rate, reset to its initial value of 0.005 between each series. We use the mini-batch gradient descent method, with the Adam optimizer and mini-batches of size 1 024, to minimize the mean absolute error (MAE).

4 Data generation

In order to train a neural network to predict the heat flux q knowing the Knudsen number ε, the density ρ, the mean velocity u and the temperature T, we need to generate a *training dataset*. This dataset must meet two criteria. First, it must be a *labelled* dataset, *i.e.* a dataset with both the input $X = (\varepsilon, \rho, u, T)$ and the expected output $Y = q$. The expected output is the "correct" output of the given input, from which we want the network to interpolate, or *generalize*, for new inputs. Then, this dataset must contain as much diversity as possible, for the interpolation to be usable in many different situations.

Consequently, the generation of the training dataset relies on two key ingredients. First, a solver for the kinetic model that can reliably estimate the distribution of particles in the phase space, from which can be derived all the fluid quantities ρ, u, T and q. Second, a mechanism to produce a variety of distributions that can be given to the solver as initial solutions. This whole process of data generation is illustrated in Figure 3 and described in more details in this section ; except for the solver for the kinetic model that is not specific to our work and is described in appendix A.

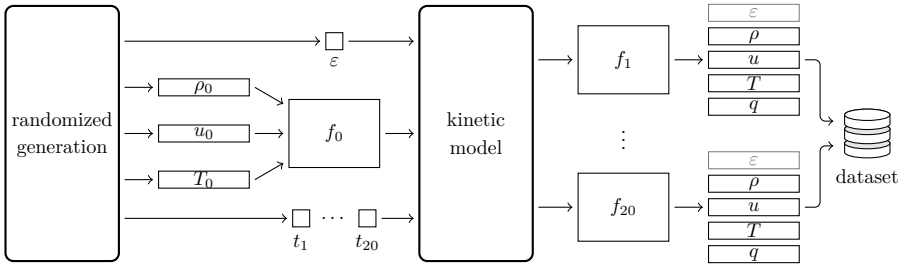

Figure 3: Scheme of the data generation process for one simulation with the kinetic model. With one Knudsen number and one initial solution, we produce 20 entries in the dataset.

4.1 Global description

Let us first get a glance at the global process illustrated by Figure 3. To begin with, a randomized process allows us to produce a Knudsen number ε, an initial solution f_0, and a *recording time* t. These three elements are given to the kinetic model that computes the evolution of the distribution up to t. The resulting distribution f is then used to compute the density ρ, the mean velocity u, the temperature T and the heat flux q of the system. On top of these four quantities, we add the Knudsen number ε as a constant vector of the same size as the other four, so that it can be used as input by the network. This could also allow us to generalize our method to a problem with a Knudsen number that varies in space. In the end, these five vectors $(\varepsilon, \rho, u, T; q)$ form an *entry* in the dataset.

Actually, in order to capitalize on the simulations that can be computationally expensive —especially in higher dimensions—, we decide to produce several entries with each simulation. For a given Knudsen number and a given initial solution, we generate not one but 20 different recording times $t_1 < \cdots < t_{20}$, and compute the distribution at each one of these times, respectively f_1, ..., f_{20}. From each one of these distributions can be derived ρ, u, T, q, to which we prepend ε to form 20 different entries to be stored in the dataset. Thus, for the cost of one simulation up to t_{20}, we produce 20 different entries corresponding to different intermediate steps in the simulation.

To build a whole dataset, we repeat this process with 100 different values of ε, and with 5 different randomly selected initial solutions for each one of these ε. Thus, we get a dataset with 10 000 entries, based on 500 different initial solutions. We produce two such datasets: a training dataset used to train the neural networks, and a test dataset used to measure and compare the performance of the trained networks on new data.

4.2 Random initial conditions

The initial solution f_0 is a Maxwellian as in (7), with density ρ, mean velocity u and temperature T randomly generated as partial Fourier series under the form

$$\alpha \times \left(\frac{a_0}{2} + 0.5 \sum_{n=1}^{N} (a_n \cos(nx) + b_n \sin(nx)) \right), \quad x \in [0, 2\pi].$$

We set N to 20 and we randomly generate the a_n and b_n coefficients for $n \geqslant 1$ according to a uniform distribution on $\left[-\frac{1}{n}, \frac{1}{n}\right]$. This choice for the a_n and b_n coefficients set the high frequencies to low amplitudes, resulting in more regular functions. Then, the choice for α and a_0 depends on the function to be generated. For the density, $\frac{a_0}{2} = 1$ and $\alpha = 1$. For the temperature, $\frac{a_0}{2} = 1$ and α is chosen uniformly in $[0.1, 1]$. For the mean velocity, $\frac{a_0}{2}$ is chosen uniformly in $[-1, 1]$, and α is chosen so that the maximum Mach number

$$\max_{x \in [0,2\pi]} \frac{|u(x)|}{\sqrt{2T(x)}},$$

falls uniformly in $[10^{-4}, 5 \cdot 10^{-1}]$ in logarithmic scale. We also make sure that the density and temperature generated are positive functions. Possible functions generated with this process are given Figure 4. For their discrete form, these functions are sampled with $N_x = 1\,024$ points to get vectors, that are turned into a discrete Maxwellian our solver can work with, with $N_v = 141$ points is velocity.

4.3 Random Knudsen numbers and recording times

In this work we focus on Knudsen numbers in the range $[\varepsilon_{\min}, \varepsilon_{\max}] = [0.01, 1]$. First because below 0.01 the fluid model with the Navier-Stokes estimation is already very accurate, and also because we do not want a range too wide. Regarding the distribution of the values in this interval, we want a decent amount of entries with ε really close to 0.01, but we do not want too few entries for ε close to 1. As a result, a uniform or logarithmic distribution over the interval $[0.01, 1]$ is not satisfying. Instead we opt for a uniform distribution for $\sqrt{\varepsilon}$ in the interval $[\sqrt{\varepsilon_{\min}}, \sqrt{\varepsilon_{\max}}]$. This allows us to have about one fourth of the Knudsen numbers between 0.01 and 0.1, and three fourths between 0.1 and 1. Also we use a deterministic distribution for the train dataset in order to have a nice coverage of the interval, while we use a random distribution for the test dataset in order to test the ability of the trained networks to generalize to new Knudsen numbers.

For the recording times $t_1 < \cdots < t_{20}$, we do not want t_1 to be too close to zero as the initial maxwellian state always has a heat flux of zero and does not carry much physical information, and we do not want t_{20} to be too big so that the simulations do not last too long. For these reasons we choose 20 times uniformly in the interval $[0.1, 2]$, that are then sorted and given successively to the solver for the kinetic model.

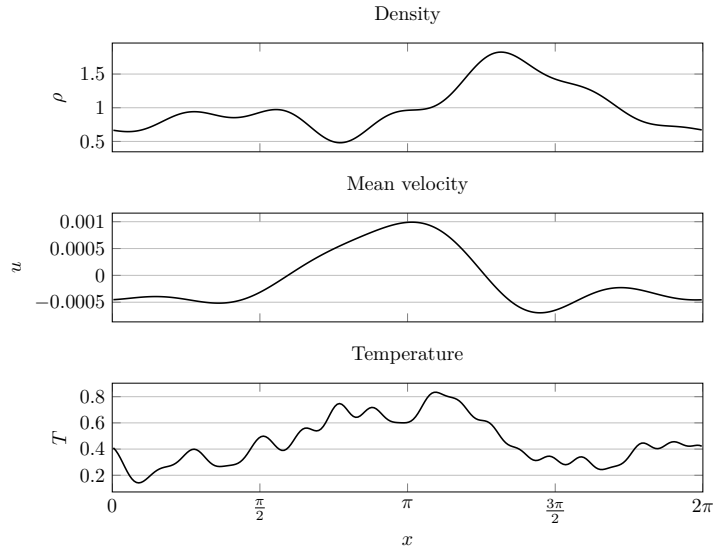

Figure 4: Possible functions ρ, u and T generated with the process described in Section 4.2.

4.4 Computing of the fluid quantities

Once the Knudsen number, the initial solution and the times are generated, a simulation is computed with the numerical method described in appendix A. It relies on a discretization of the phase space

$$(x_i, v_j) \quad , \quad i \in \{1, ..., N_x\},\ j \in \{1, ..., N_v\},$$

and results in an approximation $(f_{i,j})$ of the real distribution f:

$$f_{i,j} \simeq f(x_i, v_j).$$

From this distribution can be derived the density ρ, mean velocity u, temperature T and heat flux q needed for the dataset, as mentioned in Section 2.1. In their discrete form, these quantities are vectors computed as follows: for any $1 \leqslant i \leqslant N_x$,

$$\rho_i = \sum_{j=1}^{N_v} f_{i,j}, \quad \rho_i u_i = \sum_{j=1}^{N_v} v_j f_{i,j}, \quad \rho_i T_i = \sum_{j=1}^{N_v} (v_j - u_i)^2 f_{i,j}$$

and

$$q_i = \sum_{j=1}^{N_v} (v_j - u_i)^3 f_{i,j}.$$

For our datasets, we use a discretization with $N_x = 1\,024$ points in space, and $N_v = 141$ points in velocity.

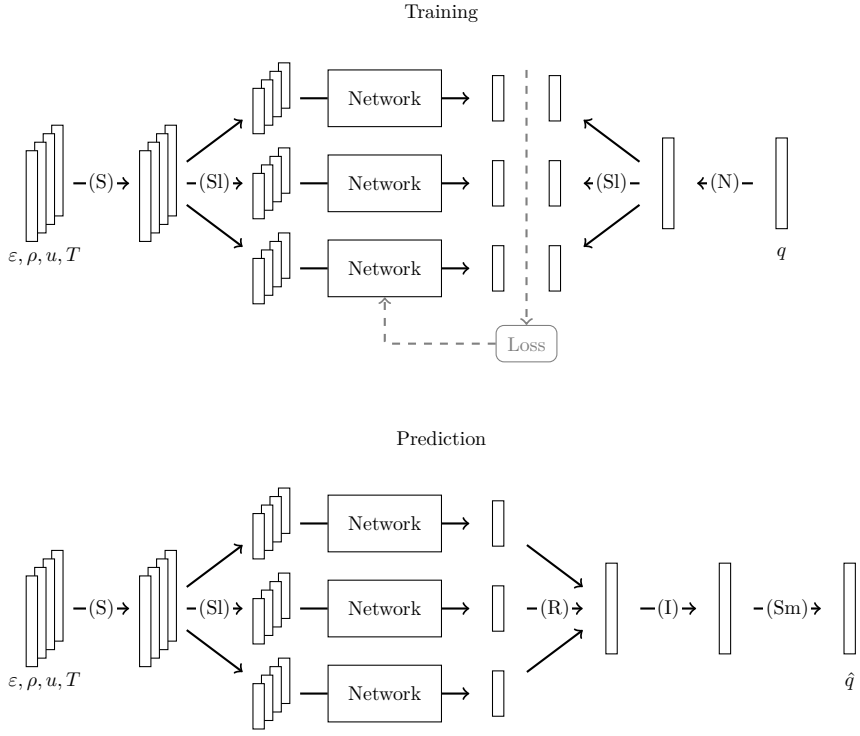

Figure 5: Scheme of the whole data processing for the training and the prediction processes respectivily. The different operations are (S) standardization, (Sl) slicing, (N) normalization, (I) inverse normalization, (R) reconstruction and (Sm) smoothing.

5 Data processing

In this section we describe how the data is processed on both sides of the neural network, both for the training process and the prediction process. At this point we have generated some data with the kinetic model, and want to process it for a neural network that takes as input a 1D signal with four *channels* —the Knudsen number, the density, the mean velocity and the temperature—, and outputs a 1D signal with one channel —the heat flux. This data processing consists in the transformations described below and summarized in Figure 5. In the remainder of this section, we use the word *standardized* to describe a quantity with mean zero and unit variance, and *normalized* a quantity with norm one or that has been applied the transformation described in Section 5.2.

5.1 Input standardization

Neural networks are known to be easier to train with standardized inputs, where all channels have the same order of magnitude. Consequently, we standardize the inputs channel by channel, by removing the mean and scaling to unit variance the whole training dataset. For example for the density ρ, if (ρ^k) is the family of densities of the whole training dataset, this standardization applied on a given entry ρ^{k_0} can be written

$$\rho^{k_0}_{\text{standard}} = \frac{\rho^{k_0} - \underset{k,i}{\text{mean}}\ \rho^k_i}{\underset{k,i}{\text{std}}\ \rho^k_i},$$

where *mean* is the empirical mean and *std* the empirical standard deviation.

Since the neural network is trained with this standardized data, it is designed to work only with inputs that received that specific standardization: after the training, any new input given to the neural network must be applied that very same standardization. In particular, the means and standard deviation computed in the training dataset have to be stored with the neural network, in order to be available later to make predictions for the fluid model.

5.2 Output normalization

To prevent the outputs in our datasets from being smaller than the typical error of our neural network, we choose to train the network to predict normalized output. Otherwise, the network would produce pure noise when trying to predict them, which turns to be problematic when used by the fluid model, especially regarding its stability. Another solution would be to use a cost fonction measuring a relative error instead of an absolute one but it turns out to be less efficient. Let us describe how we proceed.

The main issue when training the network with normalized output is the reversibility of the normalization. Since in the end we want to predict a heat flux, and not a normalized heat flux, we need to be able to apply an inverse normalization at the output of the neural network. As a consequence, this transformation cannot use any knowledge on the heat flux to be predicted, that would be available in the labelled training dataset, but would not in a real case scenario. For this reason, we choose to normalize the heat flux with its estimation given by the Navier-Stokes approximation (9), and that can be computed from the Knudsen number, the density and the temperature, all available in a real case scenario.

But using an estimation of the heat flux brings another concern: when this estimation is way off, the normalization (or inverse normalization) would distort the information a lot. And this is expected to happen, as the Navier-Stokes approximation tends to greatly overestimate big heat fluxes with Knudsen numbers in the range we consider. To prevent this from happening, we choose to normalize heat fluxes only when the corresponding Navier-Stokes estimation is

below a given threshold θ, that we set to 0.1. This threshold is also a way of only addressing the heat fluxes that were problematic in the first place.

Thus, for a given input $(\varepsilon^{k_0}, \rho^{k_0}, u^{k_0}, T^{k_0})$ in the training dataset with expected output q^{k_0}, the normalization we use can be written

$$q^{k_0}_{\text{norm}} = \begin{cases} q^{k_0} \times \dfrac{\theta}{q^{k_0}_{NS}}, & \text{if } 0 < q^{k_0}_{NS} \leqslant \theta, \\ \\ q^{k_0}, & \text{otherwise,} \end{cases}$$

with

$$q^{k_0}_{NS} = \max_{i=1,\dots,N} \left| \frac{3}{2} \varepsilon^{k_0} \rho^{k_0}_i (\partial_x T)^{k_0}_i \right|,$$

where i refers to the index of the vectors and the spatial derivative is approximated with a centered finite difference formula. Note that we choose not to normalize at all heat fluxes such that $q_{NS} = 0$. Once trained to predict these normalized heat fluxes, the neural network must be followed by an inverse normalization when used to make predictions of non normalized heat fluxes. This inverse normalization simply reads

$$q^{k_0} = \begin{cases} q^{k_0}_{\text{norm}} \times \dfrac{q^{k_0}_{NS}}{\theta}, & \text{if } 0 < q^{k_0}_{NS} \leqslant \theta, \\ \\ q^{k_0}_{\text{norm}}, & \text{otherwise.} \end{cases} .$$

5.3 Slicing and reconstructing

In order to be usable with meshes of different sizes, the neural network is designed to work with only one portion of the signal at a time as input, and to return the corresponding portion as output. As a consequence, each signal given to the network has first to be sliced into several *windows* of the right size. For the training process, both the input and the expected output have to be applied the same slicing. For the prediction process, the input has to be sliced into windows, and the resulting predictions have then to be aggregated to reconstruct a complete signal.

In one case or the other, we slice the signal into overlapping windows. The overlapping serves two purposes. For the training process, it allows us to artificially increase the amount of data we can give to the network from the training dataset. Though it introduces redundancy, such data augmentation is known to help prevent overfitting and improve the accuracy of the network [36]. For the prediction process, we use this redundancy introduced by the overlapping when reconstructing the output signal, to provide a first smoothing process and to dismiss the defects on the borders of the predictions.

These defects come from the necessary padding in our V-Net architecture, that can cause oscillations of big amplitudes at the ends of the predictions. As a consequence, each window must have a *margin* that will be ignored when

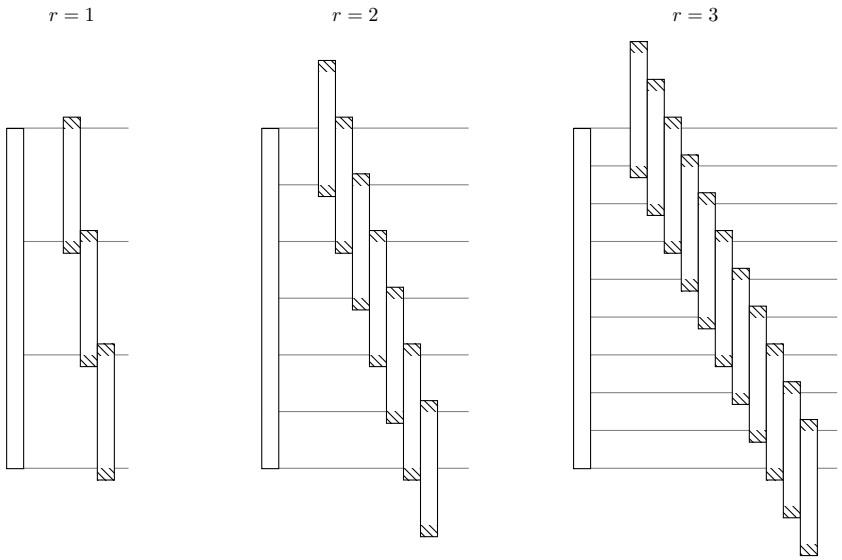

Figure 6: Slicing of a 1D signal into overlapping windows with different redundancy parameters r. The original signal is represented vertically on the left, and the windows are on the right. The hatched parts of each window are the margins that are ignored when reconstructing the signal. The windows that exceed the span of the original signal are completed by periodicity.

reconstructing the signal. We set this margin to 10% of the output on each side, leaving 80% actually useful for the reconstruction, later referred to as the *useful part*. This observation on its own is enough to require some overlapping between the windows to reconstruct the whole signal, but we introduce even more overlapping. We slice the original signal such that each one of its points is found in a fixed number r of windows, margins aside. We call this number the *redundancy parameter*. The bigger it is, the bigger the overlapping, and the more windows are produced. This slicing is illustrated Figure 6.

For the reconstruction of an entire signal from the predictions of the network, we get rid of the margins and multiply the useful parts by the kernel

$$\frac{1}{n}\left(\cos\left(\frac{2\pi}{L_u}x - \pi\right) + 1\right), \quad x \in [0, L_u]$$

where r is the redundancy parameter and L_u the length of the useful part in the physical space. We then sum up the resulting windows. This kernel has the property to sum up to one when duplicated every $\frac{L_u}{r}$ (see Figure 7). As a consequence, each value in the reconstructed signal is a weighted mean of the r values predicted by the neural network, which results in a somewhat smooth estimation of the heat flux.

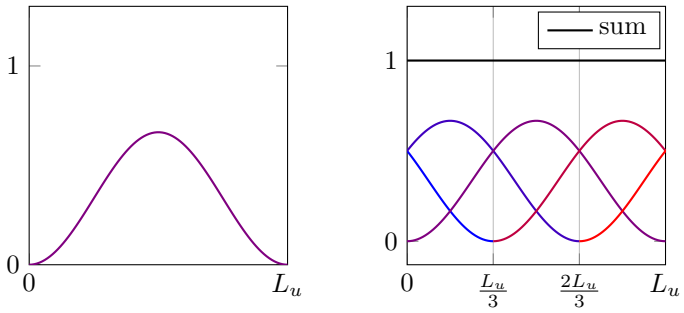

Figure 7: Kernel used for the reconstruction of an entire signal from the windows predicted by the neural network, with the redundancy parameter $r = 3$.

5.4 Smoothing of the output

At this point we have a process that allows us to estimate the heat flux q with the neural network, and that can be used in a fluid model. But as it is, even with the smoothing provided by the aggregation of different predictions as described in the previous section, this estimation can show some small irregularities. In the fluid model, these irregularities on q are amplified by the computation of $\partial_x q$, and can cause the instability of the hyperbolic system. This instability cannot be prevented by refining the time discretization, we have to control the irregularities in the estimated heat flux. To do so, we smooth the reconstruted heat flux thanks to a convolution with a gaussian kernel. This kernel with standard deviation σ reads

$$w : t \in [-3\sigma, 3\sigma] \mapsto \frac{e^{-\frac{1}{2}\frac{t^2}{\sigma^2}}}{\int_{-3\sigma}^{3\sigma} e^{-\frac{1}{2}\frac{t'^2}{\sigma^2}}\, dt'},$$

and the smoothed heat flux is

$$\tilde{q}(x) = \int_{-3\sigma}^{3\sigma} q(x+t)w(t)\, dt.$$

We show in Section 6.2.4 that σ can be chosen relatively large ($\sigma \simeq 0.05$) without impacting the accuracy of the estimation, and has to be for the method to be stable.

6 Results

This section is divided in three parts: first we introduce the results regarding the neural network only, then when it is used in the fluid model, and finally how the whole model performs in configurations that differ from the one used for the training. Unless otherwise stated, the hyperparameters of the neural network and the training are those presented in section 3.3.

6.1 Accuracy of the network

In this section we show numerical results of the neural network, independently of its use with the fluid model. To this end we use the test dataset with entries $(\varepsilon, \rho, u, T; q)$ computed with the kinetic model, or directly compare the predictions with a simulation of the kinetic model. To measure the accuracy of the network on a given entry, we use the relative error in norm L^2

$$\frac{||q - \hat{q}||_2}{||q||_2},$$

where $\hat{q}$ is the prediction of the neural network with inputs $(\varepsilon, \rho, u, T)$. We also observed the relative error in norm L^∞, but it was essentially the same, that is why we focus on the L^2 norm in what follows.

Unless stated otherwise, the results given use a redundancy parameter of 2 and a smoothing of $\sigma \simeq 0.06$. This last choice is motivated by the results of Section 6.2.4.

6.1.1 Comparison with the Navier-Stokes estimation

Figure 8 shows some examples of predictions, compared to the real heat flux and the estimation of Navier-Stokes (9). The first example illustrates how the Navier-Stokes estimation tends to perform better on data with small Knudsen numbers. The second and third example show how on the contrary it very often overestimate the heat flux when the Knudsen number increases.

The relative errors of both the network and the Navier-Stokes estimation over the whole test dataset are summarized in the histogram Figure 9. On this dataset, the overall error of the neural network is about an order of magnitude below that of the Navier-Stokes estimation.

Figure 10 gives a closer look at these errors and highlights two important factors in the performance of the network over the Navier-Stokes estimation: the Knudsen number and the norm of the real flux. The first scatter plot (Fig. 10, left) shows that the error of the Navier-Stokes estimation heavily depends on the Knudsen number ε, which is not the case for the neural network predictions, except for small values. This is even better illustrated by Figure 11, that uses the same data but where the error is not set in logarithmic scale. The second scatter plot (Fig. 10, right) shows that there is a strong correlation between the norm of the real heat flux and the relative error of the network: the smaller the heat flux, the bigger the error of the network. This could be explained by a lack of small heat fluxes in the training dataset or by the normalization we chose for small heat fluxes. In any case, it should not be that big of a problem when used with the fluid model, as smaller heat fluxes have a smaller impact on the result, at least as long as they are quite smooth.

It can also be interesting to look at the evolution of the error during a simulation, to get a better idea of what will happen when used with the fluid model. To do so, we choose 50 random initial solutions and 50 random Knudsen numbers in the same way we did to build the test dataset, and run the kinetic model

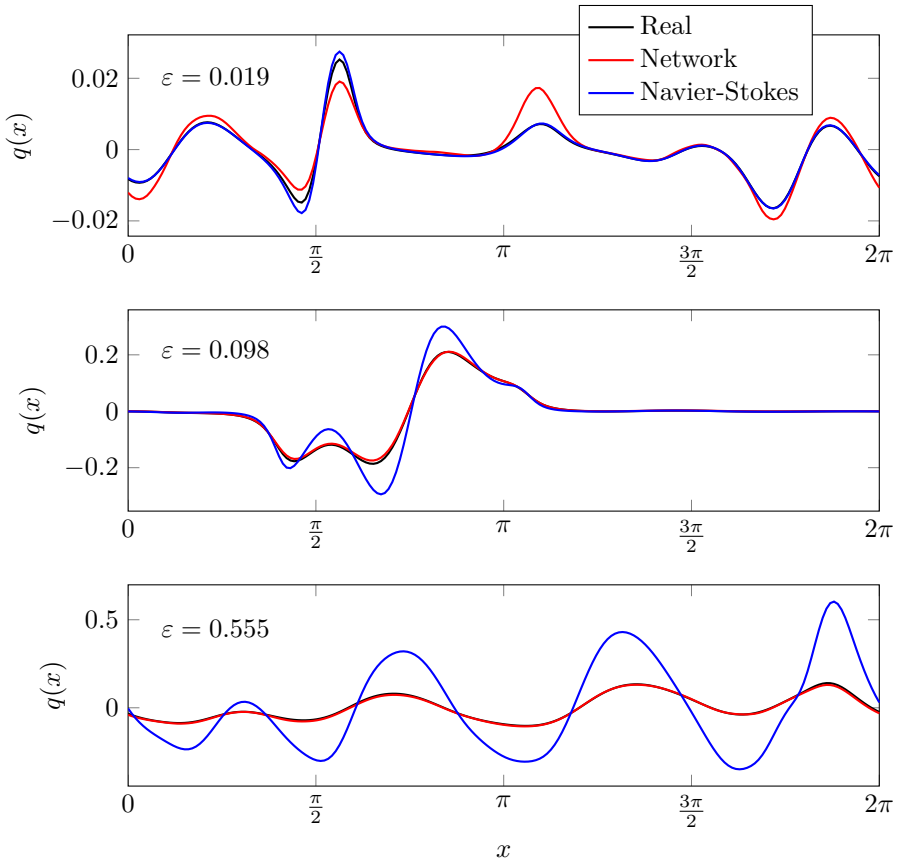

Figure 8: Three examples of predictions.

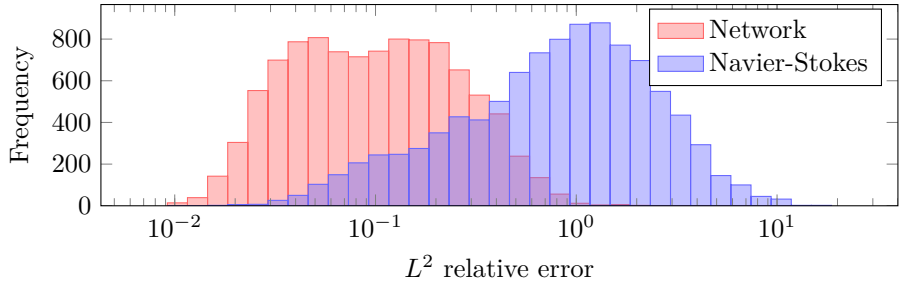

Figure 9: Distribution of the relative errors of the neural network and the Navier-Stokes estimation over the test dataset (10, 000 predictions).

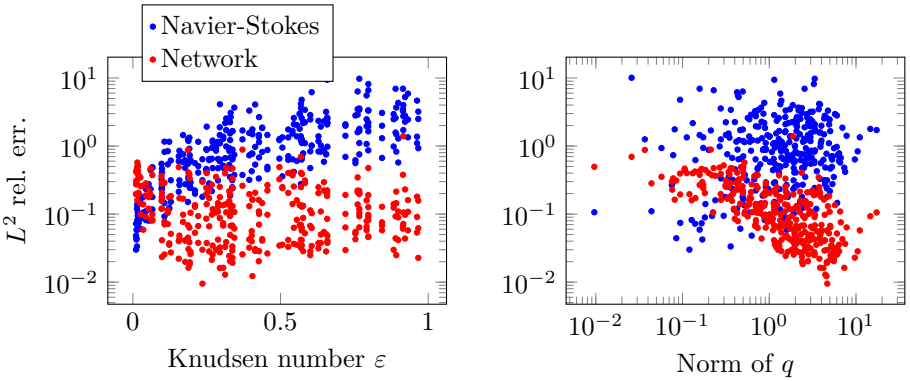

Figure 10: Error of the Navier-Stokes estimations and of the network on the test dataset, depending on the Knudsen number and the L^2 norm of the real heat flux. Each dot corresponds to one entry in the test dataset. For better clarity, only one 31^{th} of the data is shown here.

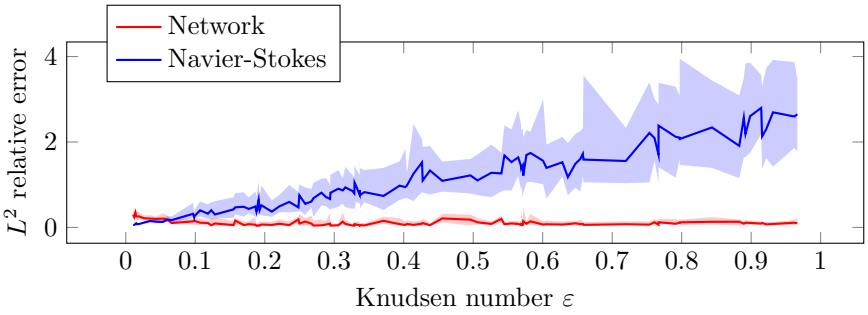

Figure 11: Relative error of the Navier-Stokes estimations and of the network on the test dataset, depending on the Knudsen number. The line represents the median of the error over the 100 entries of the test dataset for each epsilon, and the coloured area the interquartile interval.

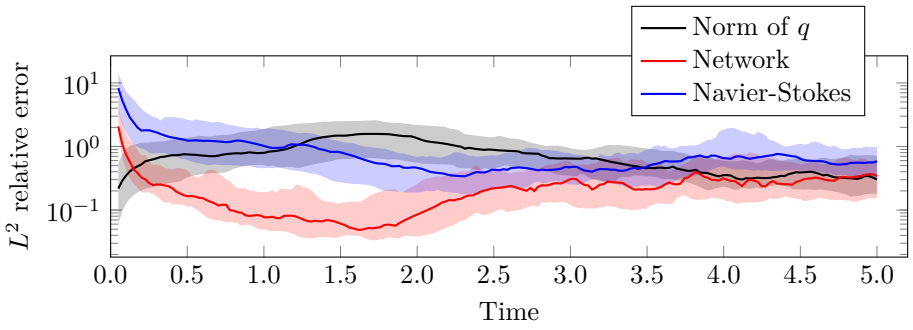

Figure 12: Norm of the real heat flux and relative errors of the predicted heat flux and the Navier-Stokes estimation throughout simulations. Median and interquartile interval over 50 simulations.

with these data. At regular times, the heat flux of the kinetic model is compared to the heat flux the network would have predicted and to the Navier-Stokes estimation. Figure 12 shows the statistical results of this experiment. We observe that the network has a significant advantage over the Navier-Stokes estimation during the first 2.5 time units, before decreasing to a smaller advantage. This result can be explained by the fact that the heat flux is bigger during the first three time units (the dissipation seems decrease the size of q in time), and by the strong correlation between the norm of the heat flux and the error of the network, also visible in this figure.

6.1.2 Hyper-parameters of the neural network

In this section we show the results that motivated the choice for some hyper-parameters of the V-Net. We focus on the number ℓ of levels, the depth d, and the size p of the kernels of the convolutions. Figure 13 compares the median relative error of V-Nets with different sets of hyper-parameters. For these results the window size was set to 256, but we later decided to set it to 512 as it gave slightly better results.

We observe that all hyper-parameters do not affect the performance of the V-Net in the same way. For instance, decreasing the size of the kernels significantly decreases the accuracy of the trained network, while not decreasing the number of parameters as much. On the other hand, decreasing the depth allows to decrease the number of parameters with a very small effect on the accuracy. In the remainder of this paper we work with the "l5,d4,p11" architecture, as it seems to be a good compromise between the number of parameters and the accuracy of the network.

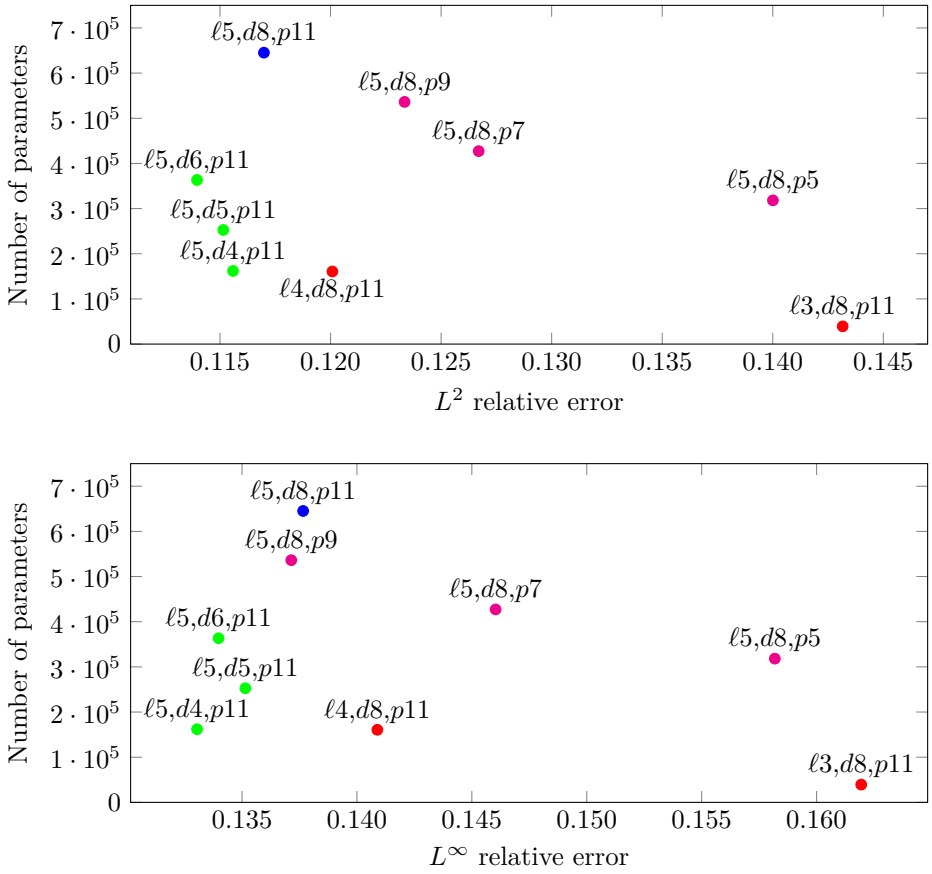

Figure 13: Median relative error and number of learnable parameters for V-Nets with different hyper-parameters : (ℓ) number of levels, (d) depth, and (p) size of the kernels. Points sharing the same color represent networks that differ by only one hyper-parameter.

6.2 Fluid model with the neural network

In this section we look at how the neural network performs when used in the fluid model. We denote by "Fluid+Network" this method. The fluid model is solved using an explicit finite volume method as presented in appendix B.1. We compare it to three others:

Kinetic : the kinetic model. It is the most accurate model and serves as a reference for the real target. The numerical method is described in appendix A.

Fluid+Kinetic : the fluid model with the heat flux from the kinetic model. It is the result we would get with a perfect neural network that makes no error, and helps to distinguish the error of the fluid model from the error of the neural network on the heat flux. The numerical method is identical to the one of the Fluid+Network method (see appendix B.1). Theoretically this model should give the same result as the kinetic model, but since the numerical schemes and viscosities are different it is not always true in practice.

Navier-Stokes : the fluid model with the Navier-Stokes estimation. It does not use the same numerical method as our model, since the formula for the heat flux requires an implicit scheme to avoid too stringent stability condition (see appendix B.2).

Our criteria to compare the fluid models is the L^2 relative error on the logarithm of the electric energy $\mathcal{E}$

$$\mathcal{E}(t) = \int_{[0,2\pi]} E(x,t)^2 \, dx,$$

compared to the kinetic model. For all the following tests we use a discretization of $N_x = 512$ points on $[0, 2\pi]$ in physical space, and $N_v = 101$ points on $[-7, 7]$ in velocity space (for the kinetic model). In particular, the data is resampled on 1 024 points at each iteration for the neural network as explained in Section 3.2. The simulations are computed up to $t = 8$.

6.2.1 Examples

Figure 14 shows examples of electric energies obtained with the four models described above. The first one uses a very small Knudsen number, and with no surprise the Navier-Stokes model performs better than the fluid model with the network. On the second example on the other hand, the Knudsen number is bigger and we observe that the oscillations are slightly shifted: Navier-Stokes shows some dispersion that the fluid model with the network does not. This dispersion often increases as the Knudsen number gets bigger as it shows on the third example. Finally the fourth one illustrates a case where the Navier-Stokes model really struggle while the fluid model with the network does pretty good.

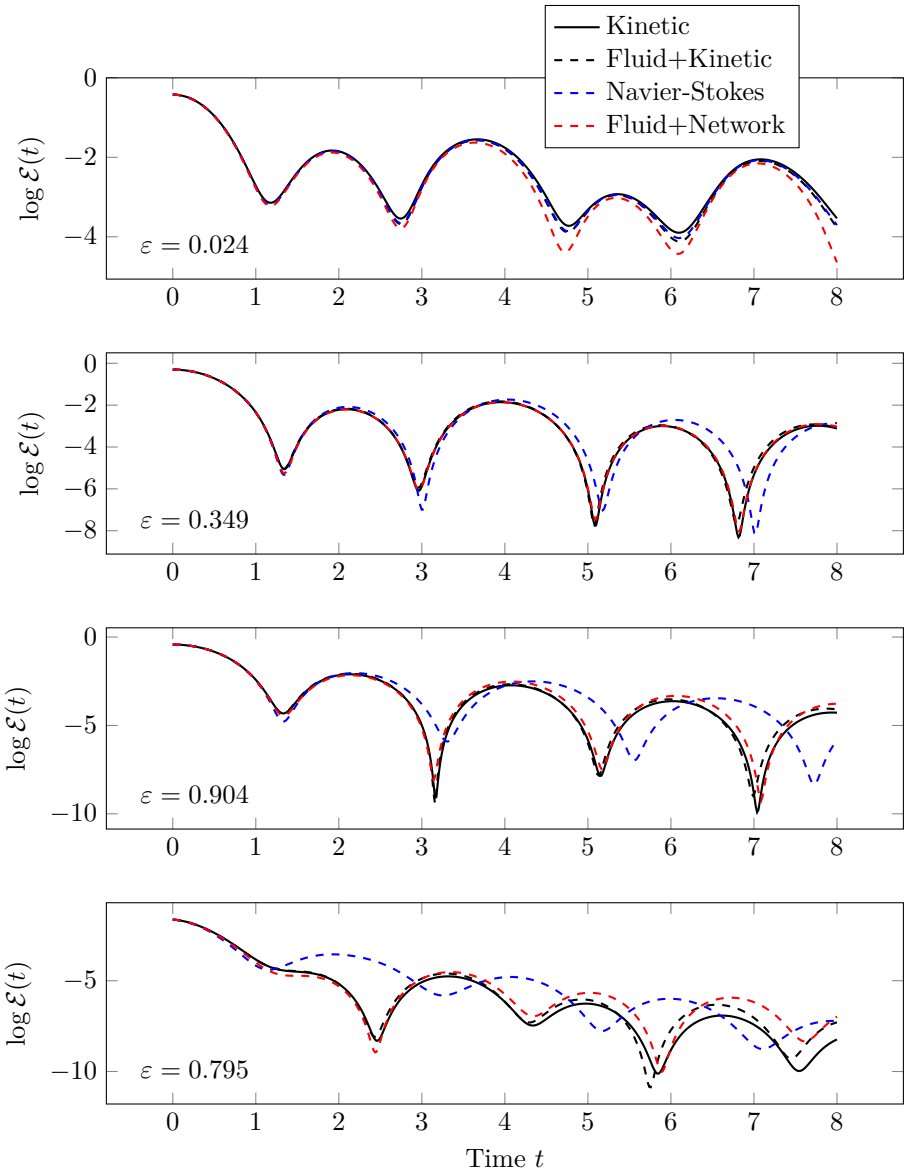

Figure 14: Examples of the evolution of the electric energy with different initial solutions and different Knudsen numbers.

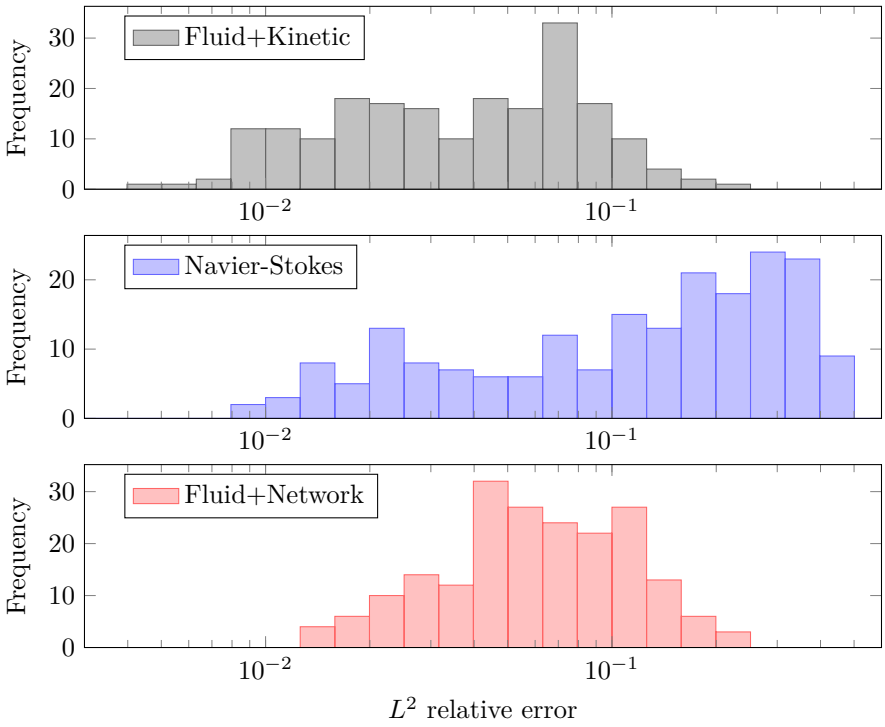

Figure 15: Distribution of the relative errors of the three fluid models on the kinetic model over 200 simulations up to $t = 8$.

6.2.2 Global performances

The L^2 relative errors of the three fluid models over 200 simulations with different initial solutions and different Knudsen numbers are summarized in Figure 15. We observe that our closure with the neural network does not reach relative errors of 0.01 or below, contrary to the closure with the real heat flux or even with the Navier-Stokes estimation. But it successfully maintains the relative error below 0.2, which is close to what the closure with the real heat flux achieves, and much better than the Navier-Stokes estimation that often exceeds 0.2. As already noted, the Fluid+Kinetic relative error should be zero but is not, due to numerical diffusion errors.

6.2.3 Influence of the Knudsen number

In Figure 16 we look at the influence of the Knudsen number on the result of the fluid models. All three seem to decrease in accuracy as the Knudsen number increases, but it is much more significant for the Navier-Stokes model. For Knudsen numbers above 0.1, not only the fluid model with the network seems to work systematically better than the Navier-Stokes model, but it is also quite

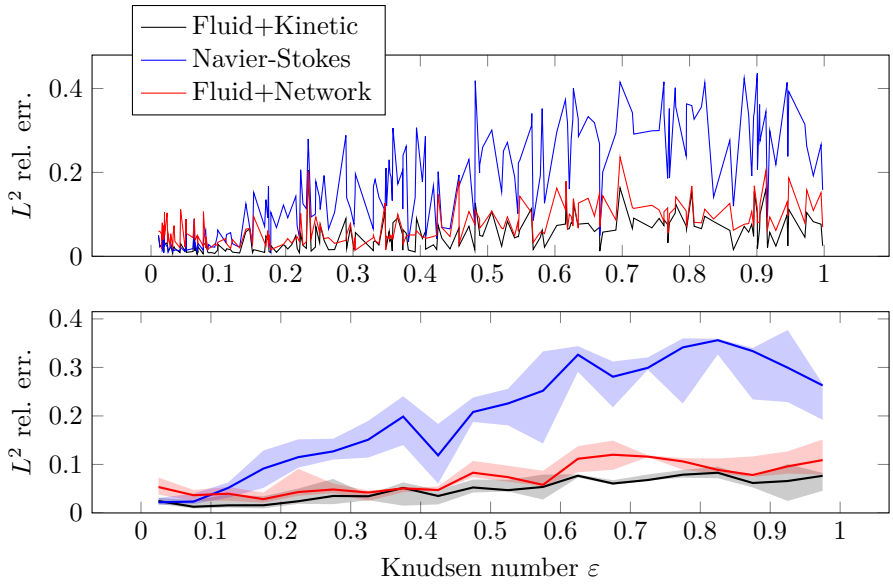

Figure 16: L^2 relative errors of the fluid models over the kinetic model depending on the Knudsen number. The first plot shows the raw data and the second shows the median and interquartile interval for 20 uniform classes of Knudsen numbers between 0.01 and 1.

close to the fluid model with the kinetic heat flux. This would seem to indicate that the network appropriately plays its role and that its error on the heat flux does not impact the whole model too much.

6.2.4 Smoothing of the prediction and stability

In this section we show the impact of the smoothing on the accuracy of the predictions, as well as its impact on the stability of the fluid model using these predictions. Figure 17 gives a visual example of how the smoothing modifies a prediction of the neural network.

To plot Figure 18, the network was used to predict the heat flux of each entry in the test dataset, and different quantities of smoothing were applied before computing the relative error. We observe that the accuracy of the resulting heat flux does not deteriorate for standard deviations lower than $\sigma \simeq 0.04$, and then slightly decreases as σ increases.

This smoothing was introduced to cope with the instability of the fluid model relying on the predictions of the network. To measure how this stability is improved by smoothing the output of the network, we choose 30 random initial solutions as we did to build the test dataset and 30 Knudsen numbers uniformly distributed between 0.01 and 1. Then we choose a set of smoothing parameters

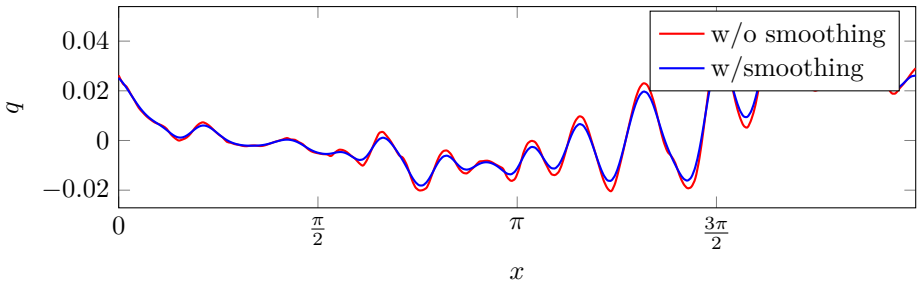

Figure 17: Example of a heat flux predicted by the neural network, with and without smoothing. The smoothing here uses $\sigma \simeq 0.05$.

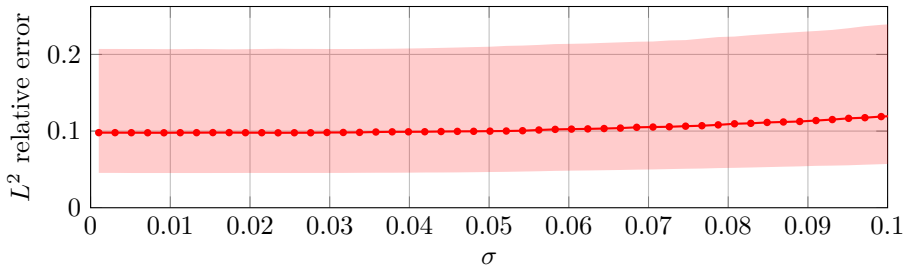

Figure 18: Relative errors of the predictions over the test dataset depending on the quantity σ of smoothing.

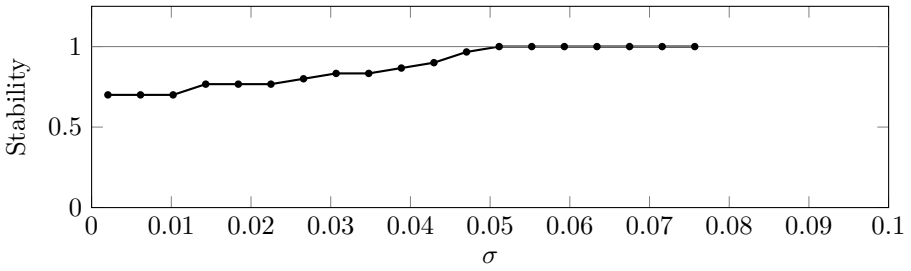

Figure 19: Proportion of simulations of the fluid model reaching $t = 3$ out of 30 simulations, depending on the quantity σ of smoothing.

σ, and for each one of these we run the 30 simulations with the fluid model, using the network with this quantity of smoothing. Finally, for each σ we look at the proportion of simulations that reached $t = 3$, from which we assume the model will remain stable. The results are shown Figure 19. It appears that all 30 simulations reach $t = 3$ for σ above approximately 0.05. We use $\sigma \simeq 0.06$ for our tests with the fluid model shown in the next section, and it appears to be enough since no simulation failed so far with this quantity of smoothing.

6.3 Using the network with different configurations

In this section we are interested in the flexibility of our approach to different configurations. First we take a look at different discretizations of space to measure the ability of the network to generalize to other resolutions than the one it has been trained with. Then we introduce discontinuities in the initial solutions to see how the fluid model with the network reacts in terms of stability and accuracy.

6.3.1 Different resolutions

The network was trained with data using 1 024 points on $[0, 2\pi]$ in space. The fact that it uses windows of size 512 would allow it to be used with any discretizations with at least 512 points, or even less if we use the periodicity to make windows of size 512. However, with initial solutions of the same form than those used to build the datasets, the change in resolution modifies the way the data is interpreted by the neural network. For instance if we sample an initial condition on 512 points instead of 1 024, the signal frequencies would be multiplied by two in the eyes of the neural network. This can result in inputs differing from the training inputs, and a decrease in accuracy is to be expected.

One way to measure this effect is to evaluate the network on the test dataset but resampled at different resolutions, to mimic the data that could be generated by the fluid model with these resolutions. We can then compute the relative errors as in Section 6.1. The results Figure 20 show that the accuracy of the network does indeed decrease as the resolution of the data moves away from its

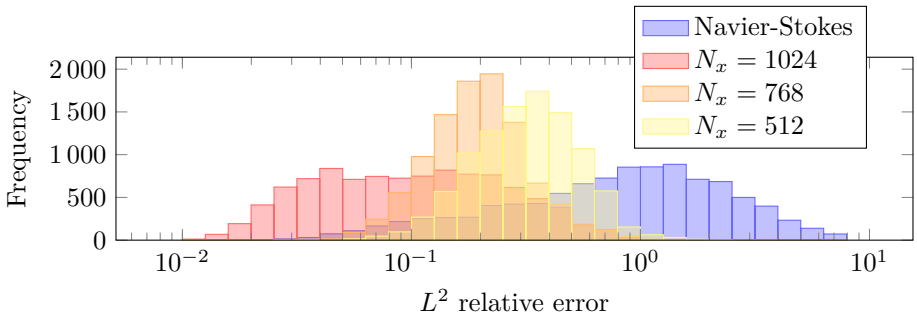

Figure 20: Distribution of the relative errors of the network on the test dataset with its original resolution, downsampled to 768 points and downsampled to 512 points.

original resolution.

To avoid this issue, if the fluid model uses a different resolution it is best to resample the data for the network to match the resolution of the training data, as this resampling operation only introduces a negligible error. However, this phenomenon shows some of the limits of the network in terms of generalization. With initial solutions that differ from the ones used to train the network —in frequency for instance—, we could expect to see a significant loss of accuracy.

6.3.2 Discontinuities

The datasets were built from simulations starting with a continuous initial solution, but using the network in the fluid model with discontinuous initial solutions could be an option. In practice, the initial discontinuities quickly fade away in this physical system, but it might be enough for the predictions of the network to cause some oscillations and the instability that goes with it.

To see if that is the case, we run multiple simulations with different Knudsen numbers and different initial solutions, similar to those used to build the dataset, except that each function (density, velocity and temperature) can be multiplied by a function

$$x \in [0, 2\pi] \mapsto \begin{cases} \frac{c}{\pi}(x - x_d) + (1 + c), & \text{if } x < x_d, \\ \frac{c}{\pi}(x - x_d) + (1 - c), & \text{if } x \geqslant x_d, \end{cases}$$

adding a discontinuity at a random location $x_d \in [0, 2\pi]$ with a random amplitude $c \in [-1, 1]$. With these initial solutions, no simulation failed to reach $t = 8$, and as can be seen Figure 21, the results are comparable to those presented in Section 6.2.

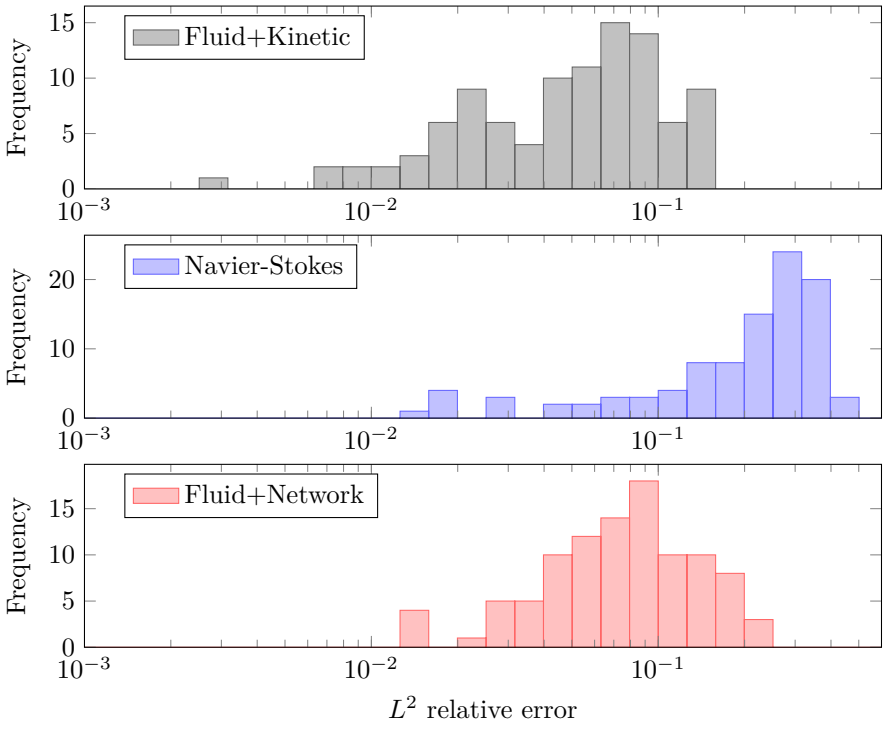

Figure 21: Distribution of the relative errors of the three fluid models on the kinetic model over 100 simulations with discontinuous initial solutions.

7 Conclusion

We construct a fluid closure for the Vlasov-Poisson dynamics based on V-net neural network and supervised learning from kinetic simulations. Slicing process of the data is introduced in order to manage meshes of different sizes. Several data processing have been also designed to improve the quality and regularity of the heat flux estimation. The numerical results show that the closure predicts the heat flux with a uniform relative error on the Knudsen interval $[0.01, 1]$, while the Navier-Stokes closure does not as expected. We also observe that the prediction is better at the beginning of the numerical simulations where the distribution function farthest from the equilibria set and so the real heat flux is larger. Surprisingly, we numerically observe that the neural network closure does not introduce instabilities when inserted in the fluid simulations, provided, however, that the outputs are regularized. Finally, the closure is quasi-optimal as the relative error between a full kinetic and a Fluid+Network closure behaves like the one between a full kinetic and a Fluid+Kinetic closure.

This work raises several issues from the numerical point of view. The first question is its efficiency in terms of computing time or its energy cost. Presently, the Fluid+Network model requires about the same computing time as the kinetic model. However, we can hope that it will be more efficient in higher dimension, as the number of computations for a V-Net with ℓ levels, depth d and kernel size p in dimension 1 is about $O(2^\ell d^2 pN)$, while it is about $O(\ell d^2 p^2 N^2)$ in dimension 2 and $O(d^2 p^3 N^3)$ in dimension 3. Compared with the $O(N^m N_v^m)$ computations required for the kinetic model where m denotes the space dimension, it increases much more slowly since $p << N_v$.Moreover, the neural network approach can greatly benefit from GPU parallelism.

The stability of the Fluid+Network closure is also an important issue. Despite the good numerical results obtained, a mathematical guaranty is lacking. Constructing neural networks closure ensuring such stability remains to be done.

Finally, we plan to apply this method to both the Vlasov-Poisson system in higher dimension and also to other closure problems arising in the MHD or gyrofluid design.

References

[1] A. Beck, D. Flad, and C. D. Munz. Deep neural networks for data-driven les closure models. *J. Comput. Phys.*, 398:108910, 2019.

[2] N. Besse, F. Berthelin, Y. Brenier, and P. Bertrand. The multi-water-bag equations for collisionless kinetic modeling. *Kinet. Relat. Models*, 2(1):39, 2009.

[3] N. Besse and P. Bertrand. Gyro-water-bag approach in nonlinear gyrokinetic turbulence. *J. Comput. Phys.*, 228(11):3973–3995, 2009.

[4] S.I. Braginskii. Transport phenomena in plasma. *Rev. plasma phys.*, 1:205, 1963.

[5] A. Brizard. Nonlinear gyrofluid description of turbulent magnetized plasmas. *Phys. Fluids B*, 4(5):1213–1228, 1992.

[6] Z. Cai, Y. Fan, and R. Li. Globally hyperbolic regularization of grad's moment system. *Communications on pure and applied mathematics*, 67(3):464–518, 2014.

[7] G.F. Chew, M.L. Goldberger, F.E. Low, and Y. Nambu. Application of dispersion relations to low-energy meson-nucleon scattering. *Phys. Rev.*, 106(6), 1957.

[8] A. Crestetto, N. Crouseilles, and M. Lemou. Kinetic/fluid micro-macro numerical schemes for vlasov-poisson-bgk equation using particles. *Kinet. Relat. Models*, 5(4):787, 2012.

[9] N. Crouseilles, P. Degond, and M. Lemou. A hybrid kinetic/fluid model for solving the gas dynamics boltzmann–bgk equation. *J. Comput. Phys.*, 199(2):776–808, 2004.

[10] P. Degond. Macroscopic limits of the boltzmann equation: a review. In *Modeling and Computational Methods for Kinetic Equations*. Birkhäuser, Boston, MA, 2004.

[11] P. Degond, G. Dimarco, and L. Mieussens. A multiscale kinetic–fluid solver with dynamic localization of kinetic effects. *J. Comput. Phys.*, 229(13):4907–4933, 2010.

[12] O. Desjardins, R. O. Fox, and P. Villedieu. A quadrature-based moment method for dilute fluid-particle flows. *J. Comput. Phys.*, 227(4):2514–2539, 2008.

[13] G Dimarco and L Pareschi. Numerical methods for kinetic equations. *Acta Numer.*, 23:369, 2014.

[14] B. Dubroca and J. L. Feugeas. Etude théorique et numérique d'une hiérarchie de modèles aux moments pour le transfert radiatif. *C. R. Acad. Sci. Paris Sér. I Math.*, 329(10):915–920, 1999.

[15] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016.

[16] K. Duraisamy, G. Iaccarino, and H. Xiao. Turbulence modeling in the age of data. *Annu. Rev. Fluid Mech.*, 51:357–377, 2019.

[17] C. K. Garrett and C. D. Hauck. A comparison of moment closures for linear kinetic transport equations: the line source benchmark. *Transport Theor. Stat.*, 42(6-7):203–235, 2013.

[18] H. Grad. On the kinetic theory of rarefied gases. *Commun. Pure Appl. Math.*, 2:331–407, 1949.

[19] G. W. Hammett, W. Dorland, and F. W. Perkins. Fluid models of phase mixing, landau damping, and nonlinear gyrokinetic dynamics. *Phys. Fluids B*, 4(7):2052–2061, 1992.

[20] G. W. Hammett and F. W. Perkins. Fluid moment models for landau damping with application to the ion-temperature-gradient instability. *Phys. Rev. Lett.*, 64:3019–3022, Jun 1990.

[21] J. Han, C. Ma, Z. Ma, and W. E. Uniformly accurate machine learning-based hydrodynamic models for kinetic equations. *PNAS*, 116(44):21983–21991, Oct 2019.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[23] P. Helluy, L. Navoret, N. Pham, and A. Crestetto. Reduced vlasov-maxwell simulations. *C. R. Mécanique*, 342(10-11):619–635, 2014.

[24] M. Junk. Domain of definition of levermore's five-moment system. *J. Stat. Phys.*, 93(5-6):1143–1167, 1998.

[25] R.J. LeVeque. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge University Press, 2002.

[26] C. D. Levermore. Moment closure hierarchies for kinetic theories. *J. Stat. Phys.*, 83(5-6):1021–1065, 1996.

[27] C. D. Levermore and W. J. Morokoff. The gaussian moment closure for gas dynamics. *SIAM J. Appl. Math.*, 59(1):72–96, 1998.

[28] G. Manfredi. Density functional theory for collisionless plasmas–equivalence of fluid and kinetic approaches. *J. Plasma Phys.*, 86(2), 2020.

[29] R. Maulik, N. A. Garland, J. W. Burby, X.-Z. Tang, and P. Balaprakash. Neural network representability of fully ionized plasma fluid model closures. *Phys. Plasmas*, 27(072106), 2020.

[30] F. Milletari, N. Navab, and S. Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571, 2016.

[31] C. Negulescu and S. Possanner. Closure of the strongly magnetized electron fluid equations in the adiabatic regime. *Multiscale Model. Sim.*, 14(2):839–873, 2016.

[32] M. Perin, C. Chandre, P. J. Morrison, and E. Tassi. Hamiltonian closures for fluid models with four moments by dimensional analysis. *J. Phys. A Math. Theor.*, 48(27):275501, 2015.

[33] N. Pham, P. Helluy, and A. Crestetto. Space-only hyperbolic approximation of the vlasov equation. *ESAIM: Proc.*, 43:17–36, 2013.

[34] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[35] J. Schneider. Entropic approximation in kinetic theory. *Esaim Math Model Numer Anal.*, 38(3):541–561, 2004.

[36] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[37] P.B. Snyder, G.W. Hammett, and W. Dorland. Landau fluid models of collisionless magnetohydrodynamics. *Phys. Plasmas*, 4(11):3974–3985, 1997.

[38] E. Sonnendrücker. *Numerical Methods for the Vlasov-Maxwell equations.* 2015.

[39] E. Tassi. Hamiltonian closures in fluid models for plasmas. *Eur. Phys. J. D*, 71(11):269, 2017.

[40] J.-X. Wang, J.-L. Wu, and H. Xiao. Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. *Physic. Rev. Fluids*, 2(3):034603, 2017.

A Numerical scheme for the kinetic model

In this appendix we describe the numerical method used to solve the Vlasov-Poisson equations (5)-(6) resulting from the kinetic model. This numerical method is used to produce data that can in turn be used by the neural network to interpolate the heat flux. It also serves as a reference to compute the error of the other methods, allowing us to compare them. For better readability, let us remind the one dimensional Vlasov-Poisson equations:

$$\partial_t f + v\,\partial_x f - E\,\partial_v f = \frac{1}{\varepsilon}(M(f) - f),$$
$$E = -\partial_x \phi, \quad \partial_{xx}\phi = \rho - \int_0^L \rho\, dx.$$

The spatial domain is given by $[0, L]$, where $L > 0$ is the spatial length. For numerical purpose, the velocity domain is restricted to the bounded interval $[-v_{\max}, v_{\max}]$. Thus we complement the equation with the following boundary conditions:

$$(\pm E)^- f = 0, \quad \text{at } v = \pm v_{\max}.$$

We consider a time discretization $(t^n)_n$ with variable time step Δt and a discretized phase space $(x_i, v_j)_{i,j}$ with constant steps Δx and Δv respectively. The

number of discretization points in space (resp. in velocity) is denoted N_x (resp. N_v). We denote by $f_{i,j}^n$ the approximation

$$f_{i,j}^n \simeq f(x_i, v_j, t^n).$$

We also use the notations $\mathbf{f}^n$ for the matrix $(f_{i,j}^n)_{i,j}$, $\mathbf{f}_i^n$ for the vector $(f_{i,j}^n)_j$ and $\mathbf{f}_j^n$ for the vector $(f_{i,j}^n)_i$.

A.1 Time discretization

To solve the Vlasov-Poisson equations over the time interval $[t^n, t^n + 1]$ we use a splitting between three stages:

1. Compute the electric field at time t^n by solving the Poisson system:

$$E = -\partial_x \phi \quad , -\partial_{xx}\phi = \frac{1}{L}\int_0^L \rho dx,$$

2. Transport the distribution function by solving the Vlasov equation over the time inverval $[t^n, t^n + 1]$:

$$\partial_t f + v\partial_x f - E\partial_v f = 0,$$

3. Update the distribution function by taking into account the collision operator:

$$\partial_t f = \frac{1}{\varepsilon}(M(f) - f).$$

The first two stages rely on the space discretization and are discussed in the next section. For the third operator with a stiff source term, we use an implicit scheme:

$$\frac{f^{n+1} - f^n}{\Delta t} = \frac{1}{\varepsilon}(M(f^{n+1}) - f^{n+1}).$$

Knowing that the fluid quantities ρ, u, T are preserved by this operator, we have $M(f^{n+1}) = M(f^n)$, so the scheme can be rewritten as:

$$f^{n+1} = f^n + \omega\,(M(f^n) - f^n), \quad \text{with } \omega = \frac{\Delta t}{\Delta t + \varepsilon}. \tag{10}$$

A.2 Spatial discretization

For the spatial discretization, we propose a method introduced in [33, 23]. First we discretize in velocity with a centered finite difference scheme. After discretization, we get the following hyperbolic system:

$$\frac{\mathbf{f}^{n+1} - \mathbf{f}^n}{\Delta t} + \Lambda\partial_x \mathbf{f}^n + EB(\mathbf{f}^n) = 0 \tag{11}$$

with Λ the diagonal matrix of velocities and $B(f^n)$ a vector given by

$$B(f^n)_j = -\frac{f^n_{j+1} - f^n_{j-1}}{2\Delta v}, \quad j \in \{2, \dots, N_v - 1\},$$
$$B_1 = -\frac{1}{2}\max(E, 0)Ef^n_1, \quad B_{N_v} = \frac{1}{2}\min(E, 0)Ef^n_{N_v}$$

The choice of the boundary terms allows to obtain a dissipative hyperbolic system and to ensure L^2 stability. Then we discretize in space the hyperbolic system (11). We use a finite volume scheme with an upwind flux:

$$\frac{\mathbf{f}^{n+1}_i - \mathbf{f}^n_i}{\Delta t} + \frac{\mathbf{f}^n_{i+\frac{1}{2}} - \mathbf{f}^n_{i-\frac{1}{2}}}{\Delta x} - EB(\mathbf{f}^n) = 0, \tag{12}$$

with $\mathbf{f}^n_{i+\frac{1}{2}} = \frac{1}{2}\Lambda(\mathbf{f}^n_{i+1} + \mathbf{f}^n_i) - \frac{1}{2}\left|\Lambda\right|(\mathbf{f}^n_{i+1} - \mathbf{f}^n_i)$.

The Poisson equation is solved using a classical finite difference scheme:

$$E^n_i = -\frac{\phi^n_{i+1} - \phi^n_{i-1}}{2\Delta x}, \quad -\frac{\phi^n_{i+1} - 2\phi^n_i - \phi^n_{i-1}}{\Delta x^2} = \rho_j - \frac{1}{N_x}\sum_i^N \rho_i. \tag{13}$$

where the density ρ_j is computed as follows: $\rho_i = \sum_{j=1}^{N_v} f_{i,j}$.

The total scheme is given by (13)-(12)-(10). The time step is chosen such as to satisfy the following stability condition:

$$\Delta t \leqslant \min\left\{\frac{\Delta x}{v_{\max}}, \frac{\Delta v}{\max_i |E_i|}\right\}.$$

Note that it is only first order accurate in order to avoid dispersive oscillations in the numerical solutions.

B Numerical scheme for the fluid models

In this section we introduce the numerical methods for solving fluid models (8). The time discretization depends on the considered closure. For the neural network based (Fluid+Network) or the kinetic one (Fluid+Network), we use an explicit scheme. For the Navier-Stokes closure (Navier-Stokes), an implicit scheme is required to avoid a too stringent stability condition. In one case or the other, the Poisson equation is solved at the beginning of each iteration in time to compute the electric field, exactly as in Eq. (13). This section therefore focuses on the fluid equations. The following schemes are classical. We briefly present them for the sake of completeness.

B.1 Explicit scheme for the neural network or the kinetic closure

Fluid equations (8) can be written

$$\partial_t \mathbf{U} + \partial_x \mathbf{F}(\mathbf{U}) = -E\mathbf{H}(\mathbf{U}), \tag{14}$$

with $\mathbf{U} = (\rho, \rho u, w)$, $\mathbf{F}(\mathbf{U}) = (\rho u, \rho u^2 + p, wu + pu + q)$, $\mathbf{H}(\mathbf{U}) = (0, \rho, \rho u)$ and where the heat flux q is given by the neural network based closure (Fluid+Network) or the one obtained from full kinetic simulations (Fluid+Kinetic). We solve it on the spatial domain $[0, L]$.

When $q = 0$ (Euler closure), equation 14 is an hyperbolic system that can be solved using a finite volume method with a local Lax-Friedrichs numerical flux and an explicit scheme in time. The hyperbolic system has characteristic speeds equal to $u, u + c, u - c$ where $c = \sqrt{3p/\rho}$ is the sound speed. When considering an additional heat flux, we propose to use the same scheme and just add a centered approximation of the heat flux in the numerical flux, as explained below.

Like for the kinetic model, we consider a time discretization $(t^n)_n$ with variable time step Δt and a discretized phase space $(x_i)_i$ with constant step Δx. We denote by U_i^n the approximation

$$\mathbf{U}_i^n \simeq \mathbf{U}(x_i, t^n).$$

The finite volume scheme results in the following formula:

$$\frac{\mathbf{U}_i^{n+1} - \mathbf{U}_i^n}{\Delta t} + \frac{\mathbf{F}(\mathbf{U})_{i+\frac{1}{2}}^n - \mathbf{F}(\mathbf{U})_{i-\frac{1}{2}}^n}{\Delta x} = -\mathbf{H}(\mathbf{U})_i^n E_i^n,$$

with

$$\mathbf{F}(\mathbf{U})_{i+\frac{1}{2}}^n = \frac{1}{2}(\mathbf{F}(\mathbf{U})_{i+1}^n + \mathbf{F}(\mathbf{U})_i^n) - \frac{S_{i+1/2}^n}{2}(\mathbf{U}_{i+1}^n - \mathbf{U}_i^n),$$

where

$$S_{i+\frac{1}{2}}^n = \max(|u_i^n| + c_i^n, |u_{i+1}^n| + c_{i+1}^n) \quad \text{and} \quad c_i^n = \sqrt{\frac{3p_i^n}{\rho_i^n}}.$$

Quantities $S_{i+\frac{1}{2}}^n$ are chosen to be larger than the maximum characteristic speed of the hyperbolic system, as u_i^n is the local speed of particles and c_i^n the local sound speed. Comparing with the classical scheme for Euler system ($q = 0$), the numerical flux for the momentum have an additional term equal to $\frac{1}{2}(q_{i+1}^n + q_i^n)$. The time step is chosen such as to satisfy the CFL stability condition:

$$\left(\max_i \, S_{i+\frac{1}{2}}^n\right) \Delta t = \frac{1}{2}\Delta x. \tag{15}$$

We refer to [25] for more details on finite volume methods.

For the Fluid+Network method, the heat flux q involved in the flux term is computed from ε, ρ, u and T at each iteration. For the Fluid+Kinetic method, it is obtained from an underlying kinetic simulation, using the same initial condition at $t = 0$. Note that the stability condition (15) does not take into account the non-zero heat flux. However, as observed in Section 6.2, this term does not result in an additional stability condition for the Fluid+Kinetic and the Fluid+Network method, provided for the latter that the heat flux is sufficiently smoothed out.

B.2 Semi-implicit scheme for the Navier-Stokes closure

With the Navier-Stokes closure $q = -\frac{3}{2}\varepsilon p \partial_x T$, the first two equations remain the same as above and are solved using the same explicit finite volume method, while the third one reads

$$\partial_t w + \partial_x(wu + pu) - \frac{3}{2}\varepsilon \partial_x(p\partial_x T) = -E\rho u.$$

To solve this equation we use the relation $w = \frac{1}{2}\rho u^2 + \frac{1}{2}\rho T$ to turn it into an equation on T, as ρ and u are known after solving the first two equations. We use a finite difference approximation for the term $\partial_x(p\partial_x T)$, and T^{n+1} can then be computed by solving the following linear system:

$$\begin{aligned}
&\frac{\frac{1}{2}\rho_i^{n+1}(u_i^{n+1})^2 + \frac{1}{2}\rho_i^{n+1}T_i^{n+1} - \frac{1}{2}\rho_i^n(u_i^n)^2 - \frac{1}{2}\rho_i^n T_i^n}{\Delta t} \\
&+ \frac{\mathbf{F}(\mathbf{U})[2]^n_{i+\frac{1}{2}} - \mathbf{F}(\mathbf{U})[2]^n_{i-\frac{1}{2}}}{\Delta x} \\
&- \frac{3}{2}\varepsilon \frac{p^n_{i+1/2}T^{n+1}_{i+1} - (p^n_{i+1/2} + p^n_{i-1/2})T_i^{n+1} + p^n_{i-1/2}T^{n+1}_{i-1}}{\Delta x^2} \\
&= -E_i^n \rho_i^n u_i^n.
\end{aligned}$$

Here the only unknown is T^{n+1}, and $\mathbf{F}(\mathbf{U})[2]$ is the third coordinate of $\mathbf{F}(\mathbf{U})$. Finally, $(w)^{n+1}$ can be computed using again the relation $w = \frac{1}{2}\rho u^2 + \frac{1}{2}\rho T$. Here the time step is chosen such as to satisfy the same CFL condition (15) as the implicit Navier-Stokes term to not introduce additional stability constraint.